Create, query and manipulate distributions with distr6 : : CHEAT SHEET

Introduction

distr6 is an object-oriented interface for probability distributions. Including distributions as objects, statistical properties of distributions, composite modelling and decorators for numerical imputation. As well as this cheat sheet, see:

- GitHub for an issue tracker and latest development branch
- CRAN for package meta-data
- The distr6 website for more complete tutorials.

R6 Classes

`Distribution` The parent class to most distr6 classes.	Distribution
`SDistribution` Class given to all probability distributions implemented in distr6.	Distribution → SDistribution
`Kernel` Class given to all kernel-like probability distributions.	Distribution → Kernel
`Decorator` Used to add or impute methods to a Distribution.	
`Wrapper` Create composite distributions by adapting class properties	
`ParameterSet` Class used to add parameters to a distribution.	

R6 Basics

$ All methods are called using dollar-sign notation	`N <- Normal$new()`
	`N$mean()`
	`N$pdf(2)`

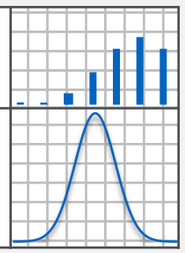
clone Objects are copied using the clone method	`N1 <- Normal$new()` `N2 <- N1$clone()`
Method chaining Call one method after another	`Normal$new()$pdf(2)`

Construct a Distribution

Each distribution has a default parameterisation, and all common parameterisations are available.

```
Binomial$new()
Binomial$new(size=5,prob=0.6)
Binomial$new(size=5,qprob=0.4)
```

```
Normal$new()
Normal$new(mean=0,sd=1)
Normal$new(mean=0,var=1)
Normal$new(mean=0,prec=1)
```



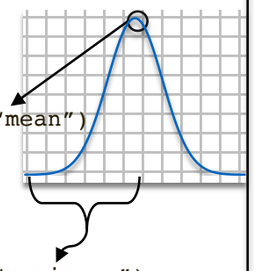
You can list all the implemented probability distributions and kernels

```
listDistributions()

listKernels()
```

Get and Set Parameters

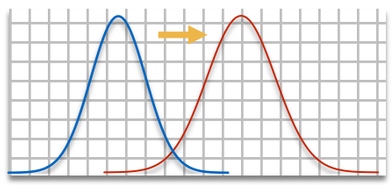
```
N <- Normal$new()
N$parameters()
N$getParameterValue("mean")
```



```
N$getParameterValue("variance")
```

Any parameter can be set, even if it wasn't used in construction. And multiple can be updated at the same time.

```
N$setParameterValue(mean = 2)
N$setParameterValue(prec = 2)
N$setParameterValue(mean = 3, sd = 3)
```



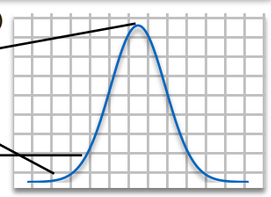
Properties and Traits

Property	Class attribute. Distribution property is dependent on parameterisation.
Trait	Class attribute. Distribution trait is independent of parameterisation.

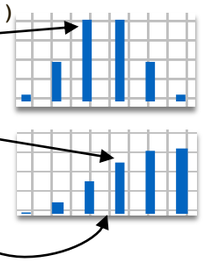
`$properties()`	`$traits()`
`$support()` values in which distribution pdf is non-zero	`$valueSupport()` discrete/continuous/ mixture
`$symmetry()` symmetric/asymmetric	`$variateForm()` univariate/multi-variate/matrixvariate
`$kurtosis()` leptokurtic/mesokurtic/ platykurtic	`$type()` Mathematical set, class *SetInterval*
`$skewness()` negative/no/positive	

Statistical Methods

```
N <- Normal$new()
N$mean()
N$variance()
N$skewness()
N$kurtosis()
N$entropy()
```



```
B <- Binomial$new(size = 5)
B$pdf(0:5)
B$cdf(0:5)
B$quantile(0.42)
B$rand(5)
```



Use ?SDistribution, ?Normal (or any other distribution) to see available methods.

S3 and Piping

distr6 uses 'R62S3' so every R6 method has an S3 dispatch available.

```
N <- Normal$new()
N$mean()                          → mean(N)
N$getParameterValue("mean")       → getParameterValue(N, "mean")
N$pdf(1:5)                        → pdf(N, 1:5)
```

Use the 'magrittr' package for method chaining and piping (%>%).

```
> N <- Normal$new()
> N$setParameterValue(sd=2)$getParameterValue("var")
```

↓ `library(magrittr)`

```
> N <- Normal$new()
> N %>% setParameterValue(sd=2) %>%
        getParameterValue("var")
```

Multivariate Distributions

Multivariate distributions are handled just like univariate distributions, except the pdf/cdf functions take multiple arguments, as do cf and mgf where available.

```
> MN <- MultivariateNormal$new(mean = c(0,0,0), cov = c(3,-1,-1,-1,1,0,-1,0,1))

> MN <- MultivariateNormal$new(mean = c(0,0,0), prec = c(3,-1,-1,-1,1,0,-1,0,1))

> MN$pdf(1, 2, 3)
> MN$cdf(1, 1, 1)
```

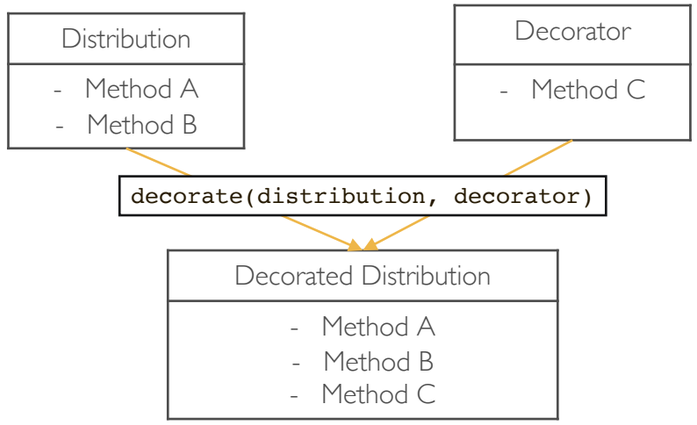
Once again vectorization is available

```
> MN$pdf(1:2, 2:3, 1:2)
> MN$cdf(c(0.45, 0.65), c(0.12, 0.99), c(0, 1))
```

Create, query and manipulate distributions with distr6 : : CHEAT SHEET

Decorators

Decorators are a design pattern (Gamma et al., 1994) used to add methods to objects.



Available Decorators

`CoreStatistics` Imputes common numeric statistical results, adds generalised expectation and moments function.

`ExoticStatistics` Adds methods for survival analysis and statistical modelling.

`FunctionImputation` Uses numerical methods to impute missing pdf/cdf/quantile/rand functions

Remember to decorate first before using a method from a decorator

```
> N <- Normal$new()
> N$survival(1)
Error: attempt to apply non-function
> decorate(N, ExoticStatistics)
> N$survival(1)
[1] 0.1586553
```

S3 methods will now work too

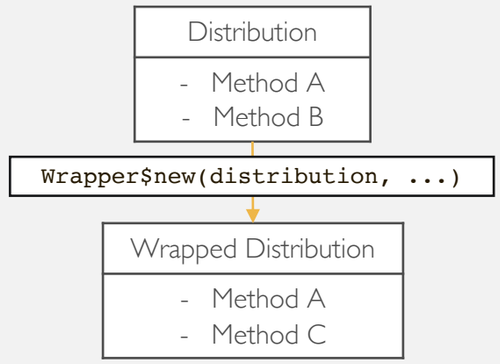
```
> N <- Normal$new(decorators = ExoticStatistics)
> pdfPNorm(N, 3, -1, 1)
[1] 0.4383636
```

Use listing to see which decorators are currently implemented.

`listDecorators()`

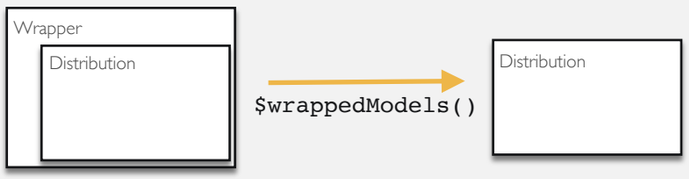
Wrappers

*Wrappers are based on the **Adapter** design pattern (Gamma et al., 1994) and are used to change the interface of an object.*



Available Wrappers

`ProductDistribution` Product of two or more distributions.	**`VectorDistribution`** Vectorizes two or more distributions.
`Convolution` Addition (or subtraction) of two distributions	**`MixtureDistribution`** Weighted mixture of two or more distributions
`HuberizedDistribution` Huberizes a distribution between limits.	**`TruncatedDistribution`** Truncates a distribution between limits.




```
> TruncatedDistribution$new(Normal$new(),
      lower = -1, upper = 1)
> MixtureDistribution$new(list(Binomial$new(),
      Normal$new()), weights = c(0.4, 0.6))
> ProductDistribution$new(list(Exponential$new(),
      Normal$new()))$pdf(1,1)
```

Use listing to see which wrappers are currently implemented.

`listWrappers()`

Custom Distributions

Custom distributions can be created using **`Distribution$new`**, *this is not the same as implementing a new* **`SDistribution`**!

```
pdf <-
function(x1)return(1/(self$getParameterValue("upper
")-self$getParameterValue("lower")))
```

The **`self`** *argument tells the object to call the method on itself.*

All **`pdf/cdf`** *methods in distr6 use* **`'x1,x2,...'`** *as their arguments*

```
cdf <- function(x1)return((x1 -
self$getParameterValue("lower")) /
(self$getParameterValue("upper") -
self$getParameterValue("lower")))
```

`ParameterSet` *is the class used for distr6 parameters.*

```
ps <- ParameterSet$new(id = list("lower","upper"),
value = c(1,10), support =
list(Reals$new(),Reals$new()), settable =
list(TRUE, TRUE))
```

The argument **`support`** *is of type* **`SetInterval`**.
See **`listSpecialSets()`**

Unique distribution **`name`** *and one-word* **`short_name`** *(ID)*

```
dist <- Distribution$new(name = "Uniform",
short_name = "unif", type = Reals$new(), support =
Interval$new(1, 10), symmetric = TRUE, pdf = pdf,
cdf = cdf, parameters = ps, description = "Custom
uniform distribution", decorators = CoreStatistics)
```

Distribution **`type`** *and* **`support`** *is of type* **`SetInterval`**.

`CoreStatistics` *decorator is optionally used to impute numeric results.*

`log` *and* **`lower.tail`** *arguments are added automatically*

```
> dist$pdf(1, log = TRUE)
[1] -2.197225
> dist$cdf(2, lower.tail = FALSE)
[1] 0.8888889
> decorate(dist, FunctionImputation)
> dist$mean()
[1] 5.5
> dist$quantile(0.42)
[1] 4.78
```

impute missing **`quantile`** *and* **`rand`** *methods*